UNIT25: SKINNED MESH – SKELETAL ANIMATION

【学習要項】
☐Skeletal animation
☐Keyframe
☐Sampling rate

【演習手順】
1．前回使用した FBX ファイル（cube.004.fbx）からアニメーションデータをロードする

2．skinned_mesh.h に animation 構造体を定義する

```
struct animation
{
    std::string name;
    float sampling_rate{ 0 };

    struct keyframe
    {
        struct node
        {
            // 'global_transform' is used to convert from local space of node to global space of scene.
            DirectX::XMFLOAT4X4 global_transform{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 };
        };
        std::vector<node> nodes;
    };
    std::vector<keyframe> sequence;
};
```

3．skinned_mesh クラスにメンバ変数(animation_clips)を追加する

```
std::vector<animation> animation_clips;
```

4．FBX シーンからアニメーションの情報を抽出する fetch_animations メンバ関数を skinned_mesh クラスに実装する

```
void skinned_mesh::fetch_animations(FbxScene* fbx_scene, std::vector<animation>& animation_clips,
    float sampling_rate /*If this value is 0, the animation data will be sampled at the default frame rate.*/)
{
    FbxArray<FbxString*> animation_stack_names;
    fbx_scene->FillAnimStackNameArray(animation_stack_names);
    const int animation_stack_count{ animation_stack_names.GetCount() };
    for (int animation_stack_index = 0; animation_stack_index < animation_stack_count; ++animation_stack_index)
    {
        animation& animation_clip{ animation_clips.emplace_back() };
        animation_clip.name = animation_stack_names[animation_stack_index]->Buffer();

        FbxAnimStack* animation_stack{ fbx_scene->FindMember<FbxAnimStack>(animation_clip.name.c_str()) };
        fbx_scene->SetCurrentAnimationStack(animation_stack);

        const FbxTime::EMode time_mode{ fbx_scene->GetGlobalSettings().GetTimeMode() };
        FbxTime one_second;
        one_second.SetTime(0, 0, 1, 0, 0, time_mode);
        animation_clip.sampling_rate = sampling_rate > 0 ?
            sampling_rate : static_cast<float>(one_second.GetFrameRate(time_mode));
        const FbxTime sampling_interval
            { static_cast<FbxLongLong>(one_second.Get() / animation_clip.sampling_rate) };
        const FbxTakeInfo* take_info{ fbx_scene->GetTakeInfo(animation_clip.name.c_str()) };
        const FbxTime start_time{ take_info->mLocalTimeSpan.GetStart() };
        const FbxTime stop_time{ take_info->mLocalTimeSpan.GetStop() };
        for (FbxTime time = start_time; time < stop_time; time += sampling_interval)
        {
            animation::keyframe& keyframe{ animation_clip.sequence.emplace_back() };

            const size_t node_count{ scene_view.nodes.size() };
            keyframe.nodes.resize(node_count);
            for (size_t node_index = 0; node_index < node_count; ++node_index)
            {
```

```
33:                 FbxNode* fbx_node{ fbx_scene->FindNodeByName(scene_view.nodes.at(node_index).name.c_str()) };
34:                 if (fbx_node)
35:                 {
36:                     animation::keyframe::node& node{ keyframe.nodes.at(node_index) };
37:                     // 'global_transform' is a transformation matrix of a node with respect to
38:                     // the scene's global coordinate system.
39:                     node.global_transform = to_xmfloat4x4(fbx_node->EvaluateGlobalTransform(time));
40:                 }
41:             }
42:         }
43:     }
44:     for (int animation_stack_index = 0; animation_stack_index < animation_stack_count; ++animation_stack_index)
45:     {
46:         delete animation_stack_names[animation_stack_index];
47:     }
48: }
```

5．skinned_mesh クラスのコンストラクタで fetch_ animations メンバ関数を呼び出す

```
float sampling_rate = 0;
fetch_animations(fbx_scene, animation_clips, sampling_rate);
```

6．skinned_mesh クラスの render メンバ関数の実装を変更する

※引数(const animation::keyframe* keyframe)を追加する

```
  1: void skinned_mesh::render(ID3D11DeviceContext* immediate_context,
  2:   const XMFLOAT4X4& world, const XMFLOAT4& material_color,
* 3:   const animation::keyframe* keyframe)
  4: {
  5:   for (mesh& mesh : meshes)
  6:   {
  7:     uint32_t stride{ sizeof(vertex) };
  8:     uint32_t offset{ 0 };
  9:     immediate_context->IASetVertexBuffers(0, 1, mesh.vertex_buffer.GetAddressOf(), &stride, &offset);
 10:     immediate_context->IASetIndexBuffer(mesh.index_buffer.Get(), DXGI_FORMAT_R32_UINT, 0);
 11:     immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
 12:     immediate_context->IASetInputLayout(input_layout.Get());
 13:
 14:     immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
 15:     immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);
 16:
 17:     constants data;
 18:     XMStoreFloat4x4(&data.world, XMLoadFloat4x4(&mesh.default_global_transform) * XMLoadFloat4x4(&world));
 19:
*20:     const size_t bone_count{ mesh.bind_pose.bones.size() };
*21:     for (int bone_index = 0; bone_index < bone_count; ++bone_index)
*22:     {
*23:       const skeleton::bone& bone{ mesh.bind_pose.bones.at(bone_index) };
*24:       const animation::keyframe::node& bone_node{ keyframe->nodes.at(bone.node_index) };
*25:       XMStoreFloat4x4(&data.bone_transforms[bone_index],
*26:         XMLoadFloat4x4(&bone.offset_transform) *
*27:         XMLoadFloat4x4(&bone_node.global_transform) *
*28:         XMMatrixInverse(nullptr, XMLoadFloat4x4(&mesh.default_global_transform))
*29:       );
*30:     }
 31:
 32:     for (const mesh::subset& subset : mesh.subsets)
 33:     {
 34:       const material& material{ materials.at(subset.material_unique_id) };
 35:       XMStoreFloat4(&data.material_color, XMLoadFloat4(&material_color) * XMLoadFloat4(&material.Kd));
 36:       immediate_context->UpdateSubresource(constant_buffer.Get(), 0, 0, &data, 0, 0);
 37:       immediate_context->VSSetConstantBuffers(0, 1, constant_buffer.GetAddressOf());
 38:       immediate_context->PSSetShaderResources(0, 1, material.shader_resource_views[0].GetAddressOf());
 39:
 40:       immediate_context->DrawIndexed(subset.index_count, subset.start_index_location, 0);
 41:     }
```

```
42:    }
43:  }
```

7．framework クラスの render メンバ関数の skinned_mesh オブジェクトの render メンバ関数の呼び出しを変更する

```
1:  int clip_index{ 0 };
2:  int frame_index{ 0 };
3:  static float animation_tick{ 0 };
4:
5:  animation& animation{ skinned_meshes[0]->animation_clips.at(clip_index) };
6:  frame_index = static_cast<int>(animation_tick * animation.sampling_rate);
7:  if (frame_index > animation.sequence.size() - 1)
8:  {
9:      frame_index = 0;
10:     animation_tick = 0;
11: }
12: else
13: {
14:     animation_tick += elapsed_time;
15: }
16: animation::keyframe& keyframe{ animation.sequence.at(frame_index) };
17: skinned_meshes[0]->render(immediate_context.Get(), world, material_color, &keyframe);
```

8．実行し、アニメーションが動作することを確認する
※サンプリングレート(sampling_rate)の値を変更し、動作の変化を確認する

9．skinned_mesh クラスのコンストラクタの引数にサンプリングレートを追加し、生成時に決定できるようにする

10．framework クラスの initialize メンバ関数で skinned_mesh コンストラクタ引数を.¥¥resources¥¥plantune.fbx に変更する
※plantune.fbx は右手系・Y 軸アップ・センチメートル単位・三角形化済み

【評価項目】
□アニメーション
□サンプリングレート